SD

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/591,986	06/12/2000	Vivek Singhal	035574-0011	5077

7590 01/05/2005

ROBERT E. KREBS
THELEN REID & PRIEST LLP
P.O BOX 640640
SAN JOSE,, CA 95164-0640

EXAMINER	
BLACKWELL, JAMES H	
ART UNIT	**PAPER NUMBER**
2176	

DATE MAILED: 01/05/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/591,986	SINGHAL ET AL.	
	Examiner	Art Unit	
	James H Blackwell	2176	

-- Th MAILING DATE of this communication app ars on th cov r sh t with the correspond nc address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *24 June 2004*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-10 and 12-36* is/are pending in the application.

4a) Of the above claim(s) *11* is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-10 and 12-36* is/are rejected.

7)☒ Claim(s) *34 and 35* is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *12 June 2000* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.

5) ☐ Notice of Informal Patent Application (PTO-152)

6) ☐ Other: _____.

DETAILED ACTION

This Office Action is in response to Amendment received 06/24/04.

Claim Objections

Claims 34 and 35 are objected to because of the following informalities: These claims, as written, depend on the apparatus of Claim 30. However, independent Claim 30 is a method claim. The Examiner has assumed that the applicant intended that Claims 34 and 35 depend on independent apparatus claim 33. Appropriate correction is required.

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 30-36 are rejected under 35 U.S.C. 102(e) as being anticipated by Challenger et al. (hereinafter Challenger, U.S. Patent No. 6,026,413).

In regard to independent Claim 30 (and similarly independent Claims 33, and 36), Challenger teaches in Fig. 1c an example of a system ... for caching dynamic Web pages. As depicted, consider a conventional Web site (100) where pages (page 1 . . . page 5) are created dynamically from one or more databases (99) and stored in one or more caches (2). ... Here, the dynamic Web pages (page 1 . . . page 5) are objects and

the underlying data (tables/records) include parts of databases (99). ... a cache manager (1) provides API's (Fig. 4), which allow an application (97) program to specify the records that a cached object depends upon. The cache manager (1) keeps track of these dependencies. Whenever an application program (97) modifies a record(s) or learns about changes to a record, which could affect the value of a complex object in a cache, the application program (97) notifies the cache manager (1) of the record(s), which has been updated (*receiving an event indicating a change in a data source*). The cache manager (1) then *invalidates or updates* all cached objects with dependencies on the record (s), which has changed (Col. 10, lines 14-24; Fig. 1c; Fig. 4; compare with Claim 30 (and similarly Claims 33, and 36), ***"... examining dependency data for the data page to determine if the data page depends on said data source; and updating the data page if the data page depends on said data source"***). Here, the cache manager examines the dependency data.

In regard to dependent Claim 31 (and similarly dependent Claim 34), Challenger teaches in Fig. 8 an example of logic for the delete_object (object_id, cache_id) (420) command. ... If in step 1107 the object (6) is found, in step 1110 the cache manager (1) deletes the objects' associated record list (11) (Fig. 3) and updates the corresponding objects lists (8) (Fig. 2). The cache manager (1) scans each record ID (12) of the record list (11) (Fig. 3) corresponding to the object (6). Note that each record ID (12) on the record list (11) has a corresponding object list (8) (Fig. 2). Pointers to object id(s) (9) (Fig. 2) corresponding to the object (6) being deleted are removed from all such object lists (8). If this results in any object list (8) becoming empty, the corresponding hash

table entry (25) is also deleted. After each element of the record list (11) is examined, it can be deleted. In step 1120, the object (6) is deleted from the object storage (4). In step 1130, the corresponding OIB (10) is deleted. Note that step 1120 can be performed concurrently with or before steps 1110 and 1130. In step 1140, the cache is unlocked and in step 1150, a status variable is returned to the application program (Col. 12, lines 44-67; Col. 131, lines 1-8; compare with Claim 31 (and similarly Claim 34), ***"... said updating includes deleting the data page from the cache if said event indicates that said data source has been deleted"***).

In regard to dependent Claim 32 (and similarly dependent Claim 35), Challenger teaches that in Fig. 1c the HTML pages, which are complex objects, are constructed from a database (99) and stored in Cache3. Each HTML page may have dependencies on one or more records which are portions of the database denoted Table1, Table2, . . , Table6. The correspondence between the tables and pages can be maintained by hash tables and record lists (19). For example, if the cache manager (1) were notified of a change to Table1 T1, it would invalidate (or update) Page1. Similarly, if the cache manager were notified of a change to Table2 T2, it would invalidate (or update) Page1, Page2, and Page3 (Col. 10, lines 25-35; compare with Claim 32 (and similarly Claim 35), ***"... said updating includes replacing the data page with an updated version of the data page if the event indicates that said data source has been updated"***).

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claims 1-2, 8-9, 12-13, 21-22, and 27-29 are rejected under 35 U.S.C. 103(a) as being unpatentable over Schloss et al. (hereinafter Schloss, U.S. Patent No. 6,249,844) in view of Challenger et al. (hereinafter Challenger, U.S. Patent No. 6,026,413).

In regard to independent Claim 1, Schloss teaches a method for dynamically parsing a digital content description of a named digital object, creating and maintaining fragment identities to facilitate caching (Col. 2, lines 36-39) that can be executed on a computing node (a Web server or Proxy server) (Col. 3, lines 61-67; compare to Claim 1, ***"A method of storing data pages at a proxy, the method comprising ..."***). Schloss also teaches in Fig. 7 an object request handler. In step (705), it is first checked whether the requested object is cached in the object cache maintained by the computing node (Col. 6, lines 31-34; compare to Claim 1, ***"... (a) receiving a data page"***). Schloss also teaches a fragment description table for tracking the object fragment identity and its description. As depicted the table (505) includes a plurality of entries (507), where each table entry (507) points to a fragment description list (510) (only one shown for ease of description). The list (510) includes one or more

description elements (520 and 525). Each fragment that maps to a given entry in the fragment description table (510) has a unique description element (520) on the fragment description list (510) of the entry. The description element includes several fields: Nlink (530); Fname (535); and Fdescription (540). The Fname (535) is the persistent name of the fragment. This name is given by the persistent name creator routine (with details depicted in Fig. 10). The Fdescription (540) is the fragment description. The Nlink (530) points to the next description element (525) that maps to the same fragment description table entry (507) (Col. 5, lines 66-67; Col. 6, lines 1-15; compare to Claim 1, ***"... (b) receiving page dependency data that contains one or more dependencies such that each dependency indicates an underlying data source which the said data page is dependent on"***). Schloss also teaches that in step (720), the computing node waits for the object requested. In step (725), after receiving the object, the object parser (with details described with reference to Fig. 8) is invoked to analyze the object description and create fragments. In step (730), the object description, which may have been modified by the object parser, is sent back to the requester. In step (735), the object cache manager is invoked to determine whether the object description (which may have been modified by the object parser) should be cached in the object cache (Col. 6, lines 37-46; compare with Claim 1, ***"... (c) storing said data page; (d) storing said page dependency data"***). Schloss fails to teach *e) receiving an event; f) determining if said event changes one of said page dependency data associated with said page data; and updating the cache by refreshing or deleting said data page if said event changes one of the page dependency data associated with said data page.*

However, Challenger teaches that in Fig. 1c the HTML pages, which are complex objects, are constructed from a database (99) and stored in Cache3. Each HTML page may have dependencies on one or more records which are portions of the database denoted Table1, Table2, ... , Table6. The correspondence between the tables and pages can be maintained by hash tables and record lists (19). For example, if the cache manager (1) were notified of a change to Table1 T1, it would invalidate (or update) Page1. Similarly, if the cache manager were notified of a change to Table2 T2, it would invalidate (or update) Page1, Page2, and Page3 (Col. 10, lines 25-35). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss and Challenger as both inventions deal with determining when and how to cache web objects. Challenger adds the benefit of providing a mechanism whereby fragments are updated or removed based on changes in metadata associated with the fragments.

In regard to dependent Claim 2, Schloss teaches an XML-like document will be used as an example of a document described using some formal language, such as a markup language. Fig. 3 shows an example of an XML-like document (Col. 4, lines 21-24; compare to Claim 2, ***"... said page dependency data are written in HTML or XML"***).

In regard to dependent Claim 8, neither Schloss nor Challenger teaches that *said page dependency data are manually encoded into a data file.* However, it would have been obvious to one of ordinary skill in the art at the time of invention to realize that one would have been able to edit a web file and manually enter the dependencies in several

ways such as placing them in the header, in view of Schloss's and Challenger's inventions because this would have provided the benefit of less processing on the file, as dependencies would not have needed to be computed.

In regard to dependent Claim 9, neither Schloss nor Challenger specifically teaches *that said data page and said data page dependency data are stored in one or more files*. However, it would have been obvious to one of ordinary skill in the art at the time of invention to write code such that the data page and its dependencies would have been saved as either one or more separate files, in view of Schloss's and Challenger's inventions, because having a single file combining both the data page and the data page dependencies would have provided the benefit of ensuring that the dependencies would not be overridden by a separate set of dependencies. Having two separate files would have provided the benefit of using one set of dependencies for multiple data pages.

In regard to dependent Claim 12, neither Schloss nor Challenger teaches that *said event is received incorporated in an event message*. However, it would have been obvious to one of ordinary skill in the art at the time of invention to flag an event in the form of a message, in view of Schloss's and Challenger's inventions, because it would have been a good way to communicate between system components using a well established mechanism like electronic mail, or that used in many web service applications, whereby events are encapsulated into a message. The benefit would have been to provide information of an event to an end user, a proxy or web manager, or a system log.

In regard to dependent Claim 13, Schloss teaches an XML-like document will be used as an example of a document described using some formal language, such as a markup language. Fig. 3 shows an example of an XML-like document (Col. 4, lines 21-24; compare to Claim 13, ***"... said event is written in HTML or XML"***).

In regard to dependent Claim 21, Schloss teaches a Least-Recently-Used (LRU) algorithm which is usually trigger invoked (Col. 8, lines 36-38; compare to Claim 21, ***"... said event came from a trigger-based event generator"***).

In regard to dependent Claim 22, neither Schloss nor Challenger specifically teaches that *said event came from a polling event generator*. However, it would have been obvious to one of ordinary skill in the art at the time of invention to use such an event generator, in light of Schloss's and Challenger's inventions, providing the benefit of having had a way to periodically scan for changes to a database and then sending out an alert signal when a change had occurred.

In regard to dependent Claim 27, Schloss teaches a fragment cache manager that uses an Least-Recently-Used (LRU) type replacement policy (Col. 8, lines 36-38; compare to Claim 27, ***"... said updating the cache involves keeping the index of URL addresses and page dependency data up-to-date"***).

In regard to independent Claim 28, Schloss teaches a method for dynamically parsing a digital content description of a named digital object, creating and maintaining fragment identities to facilitate caching (Col. 2, lines 36-39) that can be executed on a computing node (a Web server or Proxy server) (Col. 3, lines 61-67; compare to Claim 28, ***"A computer software product for us in a comput r system that executes***

program steps recorded in a computer-readable media to perform a method for enabling storing of data pages at a proxy comprising ..."). Schloss also teaches software for execution on a computer or other processor-based device. The software may be embodied on a magnetic, electrical, optical, or other persistent program and/or data storage device, including but not limited to: magnetic disks, DASD, bubble memory; tape; optical disks such as CD-ROMs; and other persistent (also called non-volatile) storage devices such as core, ROM, PROM, flash memory, or battery backed RAM (Col. 10, lines 14-22; compare to Claim 28, ***"... (a) a recordable media; (b) a program of computer-readable instructions executable by the computer to perform method steps comprising..."***). Schloss also teaches in Fig. 7 an object request handler. In step (705), it is first checked whether the requested object is cached in the object cache maintained by the computing node (Col. 6, lines 31-34; compare to Claim 28, ***"... (i) receiving a data page"***). Schloss also teaches a fragment description table for tracking the object fragment identity and its description. As depicted the table (505) includes a plurality of entries (507), where each table entry (507) points to a fragment description list (510) (only one shown for ease of description). The list (510) includes one or more description elements (520 and 525). Each fragment that maps to a given entry in the fragment description table (510) have a unique description element (520) on the fragment description list (510) of the entry. The description element includes several fields: Nlink (530); Fname (535); and Fdescription (540). The Fname (535) is the persistent name of the fragment. This name is given by the persistent name creator routine (with details depicted in Fig. 10). The Fdescription

(540) is the fragment description. The Nlink (530) points to the next description element (525) that maps to the same fragment description table entry (507) (Col. 5, lines 66-67; Col. 6, lines 1-15; compare to Claim 28, ***"... (ii) receiving page dependency data that contains one or more dependencies such that each dependency indicates an underlying data source which the said data page is dependent on"***). Schloss also teaches that in step (720), the computing node waits for the object requested. In step (725), after receiving the object, the object parser (with details described with reference to Fig. 8) is invoked to analyze the object description and create fragments. In step (730), the object description, which may have been modified by the object parser, is sent back to the requester. In step (735), the object cache manager is invoked to determine whether the object description (which may have been modified by the object parser) should be cached in the object cache (Col. 6, lines 37-46; compare with Claim 28, ***"... (iii) storing said data page; (iv) storing said page dependency data"***). Schloss fails to specifically teach *(v) receiving an event; (vi) determining if said event changes one of the said page dependency data associated with said data page; and (vii) updating the cache by refreshing or deleting said data page if said event changes said one of the page dependency data associated with said data page*. However, Challenger teaches in Fig. 7 that each fragment of a page has associated with it a number of metadata parameters that are checked whenever an event, such as a request for that fragment from a web client is received. Changes in any one or more of these parameters determine when and how data are invalidated and hence removed or updated in the cache (Col. 9, lines 11-15). Challenger also teaches in Fig. 12 that a

request may be received from a client or from another JSP in JVM 1200, which invokes the whole process for determining the cachability of a fragment dependent upon metadata associated with the fragment (Col. 12, lines 67; Col. 13, lines 1-8). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss and Challenger as both inventions deal with determining when and how to cache web objects. Challenger adds the benefit of providing a mechanism whereby fragments are updated or removed based on changes in metadata associated with the fragments (note that metadata perhaps is contained in the fragment and not separate file).

In regard to independent Claim 29, Schloss teaches proxy servers (30 . . . 33) or Web content servers (40 . . . 43). The proxy servers and Web servers can provide caching of frequently accessed Web objects to improve client access time (Col. 3, lines 61-67; compare to Claim 29, ***"A proxy server system that provides stored data files without requesting data files from the origin web server, ..."***). Schloss also teaches an Internet environment containing a client (60 ...63) connected through a network (25) to access proxy servers (30 ... 33) or web content servers (40 ... 43) (Col. 3, lines 61-65; Fig. 1; compare with Claim 29, ***"... (a) a central processing unit that can establish communication with a user computer"***). Schloss also teaches that the computing node can include: a CPU (250); a scratch pad or main memory (245) such as RAM; and persistent storage devices (260) such as direct access storage devices (DASD). The memory (245) stores the server logic 240 (with details depicted in Fig. 6) preferably embodied as computer executable code that may be loaded from

DASD (260) into memory (245) for execution by CPU (250) (Col. 4, lines 1-13; compare to Claim 29, ***"... (b) a storage device; (c) a processor connected to the storage device wherein the storage device stores: (i) at least one program component for controlling the processor; and (d) the processor is operative with said program component to ..."***). Schloss also teaches in Fig. 7 an object request handler. In step (705), it is first checked whether the requested object is cached in the object cache maintained by the computing node (Col. 6, lines 31-34; compare to Claim 29, ***"... (i) receive a data page"***). Schloss also teaches a fragment description table for tracking the object fragment identity and its description. As depicted the table (505) includes a plurality of entries (507), where each table entry (507) points to a fragment description list (510) (only one shown for ease of description). The list (510) includes one or more description elements (520 and 525). Each fragment that maps to a given entry in the fragment description table (510) have a unique description element (520) on the fragment description list (510) of the entry. The description element includes several fields: Nlink (530); Fname (535); and Fdescription (540). The Fname (535) is the persistent name of the fragment. This name is given by the persistent name creator routine (with details depicted in Fig. 10). The Fdescription (540) is the fragment description. The Nlink (530) points to the next description element (525) that maps to the same fragment description table entry (507) (Col. 5, lines 66-67; Col. 6, lines 1-15; compare to Claim 29, ***"... (ii) receive page dependency data that contains one or more dependencies such that each dependency indicates an underlying data source which the said data pag is dependent on"***). Schloss also teaches that in

step (720), the computing node waits for the object requested. In step (725), after receiving the object, the object parser (with details described with reference to Fig. 8) is invoked to analyze the object description and create fragments. In step (730), the object description, which may have been modified by the object parser, is sent back to the requester. In step (735), the object cache manager is invoked to determine whether the object description (which may have been modified by the object parser) should be cached in the object cache (Col. 6, lines 37-46; compare with Claim 29, ***"... (iii) store said data page; (iv) store said page dependency data"***). Schloss fails to specifically teach *(v) receiving an event; (vi) determining if said event changes one of the said page dependency data associated with said data page; and (vii) updating the cache by refreshing or deleting said data page.* However, Challenger teaches in Fig. 7 that each fragment of a page has associated with it a number of metadata parameters that are checked whenever an event, such as a request for that fragment from a web client is received. Changes in any one or more of these parameters determine when and how data are invalidated and hence removed or updated in the cache (Col. 9, lines 11-15). Challenger also teaches in Fig. 12 that a request may be received from a client or from another JSP in JVM 1200, which invokes the whole process for determining the cachability of a fragment dependent upon metadata associated with the fragment (Col. 12, lines 67; Col. 13, lines 1-8). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss and Challenger as both inventions deal with determining when and how to cache web objects. Challenger adds the benefit of providing a mechanism whereby fragments are updated or removed

based on changes in metadata associated with the fragments (note that metadata perhaps is contained in the fragment and not separate file).

Claims 3-7, 10, 14-20, and 26 are rejected under 35 U.S.C. 103(a) as being unpatentable over Schloss in view of Challenger and in further view of Batchelder et al. (hereinafter Batchelder, U.S. Patent No. 6,351,767).

In regard to dependent Claim 3, neither Schloss nor Challenger specifically teaches that *said page dependency data are generated by a Request-based dependency generator.* However, Batchelder teaches a URL Parser (303), which breaks the URL into different parts. The parsed URL is passed to the cache control unit (311) (Col. 7, lines 13-15). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with caching dynamic web content. Batchelder provides the benefit of specifying a URL-based event mechanism to change the status of a cache.

In regard to dependent Claim 4, neither Schloss nor Challenger specifically teaches that *said Request-Based dependency generator uses a URL request of the said data page*. However, Batchelder teaches a URL Parser (303) that breaks the URL into different parts. Specifically, commands to perform functions are added to the URL (after a question mark separator) (Col. 7, lines 12-27). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with

issues associated with caching dynamic web content. Batchelder provides the benefit of specifying a URL-based event mechanism to change the status of a cache.

In regard to dependent Claim 5, neither Schloss nor Challenger specifically teaches that *said Request-Based dependency generator uses a configuration file to generate said page dependency data*. However, Batchelder teaches that Fig. 5 depicts an example of a fragment description table for tracking the object fragment identity and its description. As depicted the table (505) includes a plurality of entries (507), where each table entry (507) points to a fragment description list (510) (only one shown for ease of description). The list (510) includes one or more description elements (520 and 525) (Col. 5, lines 66-67; Col. 6, lines 1-5). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with caching dynamic web content. Batchelder adds the benefit of tracking object fragment parameters.

In regard to dependent Claim 6, neither Schloss nor Challenger specifically teaches that *said page dependency data are generated by a script-based dependency generator*. However, Batchelder teaches that once the web page response is built by the response builder (307), it is passed to the HTML unit (305) for conversion to HTML. This HTML response is then passed to the HTTP server (206) for serving to the requesting user (Col. 7, lines 59-62). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with

caching dynamic web content. Batchelder adds the benefit of having executed commands within software instead of typing them as part of a URL.

In regard to dependent Claim 7, neither Schloss nor Challenger teaches that *said page dependency data are encoded in the said data page by a script-based dependency generator*. However, Batchelder teaches that a response builder (307) uses the parsed URL to build the response by accessing the appropriate sources (via source access unit (308)) and retrieving the appropriate "parts" to construct the response. The parts retrieved by the response builder (307) may comprise many different types, including data, forms, subforms, database design elements, calculations, etc. In other words, there is no theoretical restriction as to the type of parts comprising a web page response. These parts each have their own attributes. For instance, some parts may or may not have last modified dates associated with the part. The attributes of all of the parts used to build the response are collected and analyzed by attribute analyzer (313). The attribute analyzer (313) builds a "composite" of the attributes, the attribute composite being representative of the entire response (Col. 7, lines 43-58). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with caching dynamic web content. Batchelder adds the benefit of having had server commands and page parameters together in one file.

In regard to dependent Claim 10, neither Schloss nor Challenger teaches that *said storing of said data page at the proxy is in response to data in a configuration file.*

However, Batchelder teaches a cacheability analyzer (309) that examines the attribute composite and, if it determines that the response cannot be cached, the response is not cached. If it determines that the response can be cached, it provides an indication to the cache control unit (311), along with the response and an associated set of cache strategy indicators generated by the cacheability analyzer (309) (Col. 7, lines 64-67; Col. 8, lines 1-5). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with caching dynamic web content. Batchelder adds the benefit of enabling the determination of cacheability.

In regard to dependent Claim 14, neither Schloss nor Challenger specifically teaches that *said event came from a Request-based dependency generator.* However, Batchelder teaches a URL Parser (303), which breaks the URL into different parts. The parsed URL is passed to the cache control unit (311) (Col. 7, lines 13-15). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with caching dynamic web content. Batchelder adds the benefit of having provoked a change in the status of data pages on a server.

In regard to dependent Claim 15, neither Schloss nor Challenger specifically teaches that *said Request-Based dependency generator uses a configuration file.* However, Batchelder teaches that Fig. 5 depicts an example of a fragment description table for tracking the object fragment identity and its description. As depicted the table (505) includes a plurality of entries (507), where each table entry (507) points to a

fragment description list (510) (only one shown for ease of description). The list (510) includes one or more description elements (520 and 525) (Col. 5, lines 66-67; Col. 6, lines 1-5). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with caching dynamic web content. Batchelder adds the benefit of having provoked a change in the status of data pages on a server.

In regard to dependent Claim 16, neither Schloss nor Challenger specifically teaches that *said Request-Based dependency generator uses a URL request of the said data page*. However, Batchelder teaches a URL Parser (303) that breaks the URL into different parts. Specifically, commands to perform functions are added to the URL (after a question mark separator) (Col. 7, lines 12-27). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with caching dynamic web content. Batchelder adds the benefit of having a means to have controlled the actions of a proxy or web server.

In regard to dependent Claim 17, neither Schloss nor Challenger specifically teaches that *said Request-Based dependency generator uses a URL request of the said data page*. However, Batchelder teaches a URL Parser (303) that breaks the URL into different parts. Specifically, commands to perform functions are added to the URL (after a question mark separator) (Col. 7, lines 12-27). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of

Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with caching dynamic web content. Batchelder adds the benefit of having a means to have controlled the actions of a proxy or web server.

In regard to dependent Claim 18, neither Schloss nor Challenger specifically teaches that *said URL request includes request header information*. However, Batchelder teaches sending server-specific commands requesting actions to be taken (Col. 7, lines 12-34). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with caching dynamic web content. Batchelder adds the benefit of having a means to have controlled the actions of a proxy or web server.

In regard to dependent Claim 19, neither Schloss nor Challenger specifically teaches that *said URL request is parsed to obtain parameters.* However, Batchelder teaches a URL Parser (303) that breaks the URL into different parts. Specifically, commands to perform functions are added to the URL (after a question mark separator) (Col. 7, lines 12-27). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with caching dynamic web content. Batchelder adds the benefit of having commanded a server to execute a given action, such as opening a database.

In regard to dependent Claim 20, neither Schloss nor Challenger specifically teaches that *said event came from a script-based dependency generator*. However,

Batchelder teaches that once the web page response is built by the response builder (307), it is passed to the HTML unit (305) for conversion to HTML. This HTML response is then passed to the HTTP server (206) for serving to the requesting user (Col. 7, lines 59-62). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with caching dynamic web content. Batchelder adds the benefit to the user of not having had to remember a complicated URL by having it hard-coded.

In regard to dependent Claim 26, neither Schloss nor Challenger teaches that *said event generated by Request-Based event generator and sent to a change event evaluator*. However, Batchelder teaches that for those URLs having commands requesting a possibly cached response (i.e., ?OpenDatabase, ?OpenView, ?OpenDocument, ?OpenForm, and ?ReadForm), the cache control (311) examines the request against previously cached responses to determine whether any of the previously cached responses is appropriate for the request. It does this by comparing the parsed URL against the URLs of the previously cached responses in the cache (304). If there is not an exact match or if the URL doesn't have "cacheable" commands (e.g., ?EditDocument), the parsed URL is passed to the response builder (307) (Col. 7, lines 34-43). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Batchelder because all three of these inventions deal with issues associated with caching dynamic web content. Batchelder adds the benefit of having sent commands to the server.

Claim 23 is rejected under 35 U.S.C. 103(a) as being unpatentable over Schloss in view of Challenger, and in further view of Arlitt et al. (hereinafter Arlitt, U.S. Patent No. 6,272,598).

In regard to dependent Claim 23, neither Schloss nor Challenger specifically teaches that *said event came from a custom event generator.* However, Arlitt teaches that other known cache replacement policies may also be used for the cache (72) (Col. 7, lines 28-29). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Arlitt because all three inventions deal with issues related to changing the contents of a cache, especially when it comes to dynamic content. Arlitt adds the benefit of having had other means to alert for changes that were unique.

Claims 24-25 are rejected under 35 U.S.C. 103(a) as being unpatentable over Schloss in view of Challenger and in further view of Bourne et al. (hereinafter Bourne, U.S. Patent No. 6,584,548).

In regard to dependent Claim 24, neither Schloss nor Challenger specifically teaches that *said determination if said event changes one of the page dependency data associated with said data page is done by a change event evaluator.* However, Bourne teaches a cache coordinator, wherein the cache coordinator invalidates one or more cache entries in response to a signal (Col. 2, lines 49-50). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of

Schloss, Challenger, and Bourne as all three inventions deal with altering the status of a cache. Bourne adds the benefit of having determined when a change had taken place.

In regard to dependent Claim 25, neither Schloss nor Challenger teaches that *said determination by the change event evaluator is done by matching said page dependency data with said event*. However, Bourne teaches an ID-based invalidation process, wherein a cache entry is associated with an ID that uniquely identifies the cache entry and can optionally be associated with one or more data ids that represent the underlying data contained in the cache entry, and the ID-based invalidation process sends a signal to the cache coordinator to invalidate all cache entries that either have that cache entry ID or have been associated with a data ID when the data that the ID represents changes (Col. 2, lines 51-58). It would have been obvious to one of ordinary skill in the art at the time of invention to combine the teachings of Schloss, Challenger, and Bourne as all three inventions deal with altering the status of a cache. Bourne adds the benefit of having updated a cache.

Response to Arguments

Applicant's arguments with respect to claims 1-36 have been considered but are moot in view of the new ground(s) of rejection.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to James H Blackwell whose telephone number is 571-272-4089. The examiner can normally be reached on Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Joseph H Feild can be reached on 571-272-4090. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

James H. Blackwell
12/21/04

JOSEPH FEILD
SUPERVISORY PATENT EXAMINER